FACULTAD DE INFORMATICA

UNIVERSIDAD COMPLUTENSE DE MADRID

PROYECTO DE SISTEMAS INFORMATICOS

AceptaElRetoApp
App Android aceptaelreto.com

José A. Flores Roch
Lorenzo A. Todisco López

Curso 2014-2015

Director:

Luís Hernández Yañez

Codirector:

Marco Antonio Gómez Martín

José A. Flores Roch
Lorenzo A. Todisco López

José A. Flores Roch
Lorenzo A. Todisco López

Declaración de Conformidad

Los alumnos abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar solo con fines académicos, no comerciales, mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

Madrid, 1 de Septiembre de 2015.

José A. Flores Roch

Lorenzo A. Todisco López

Agradecimientos

En primer lugar quisiéramos agradecer a Marco Antonio Gómez Martín y Pedro Pablo Gómez Martin por toda la ayuda que nos han prestado para poder realizar este proyecto, y a Luis Hernández Yáñez por hacer que fuera posible realizarlo como trabajo de Sistemas Informáticos.

A mi familia, por todo el apoyo y empuje durante toda la carrera. A mis amigos, aquellos con los que he compartido mis años de universidad y a aquellos con los que he compartido mucho más.

José

A mi familia, por creer en mí y siempre brindarme el apoyo incondicional para seguir adelante. A mis amigos, por estar siempre presentes tanto en las malas como en las buenas.

Lorenzo

A todos ellos muchas gracias.

José A. Flores Roch
Lorenzo A. Todisco López

Resumen

¡AceptaElReto! es un almacén y juez en línea de problemas de programación en español que acepta soluciones en los lenguajes C, C++ y Java.

¡AceptaElReto App! es una aplicación desarrollada en Android que facilita a los usuarios de dispositivos móviles y tablets, navegar e interactuar de una forma cómoda y flexible con el juez en línea.

La aplicación envía y recibe información a través de servicios web. Para permitir el uso de la aplicación sin conexión a internet, hemos hecho uso de la memoria caché de los dispositivos para almacenar información necesaria para el usuario, así como soluciones parciales o completas generadas por los mismos.

Para facilitar el manejo de nuestra aplicación, la hemos desarrollado siguiendo las pautas acordadas en la guía/página de AndroidDevelopers, consiguiendo un entorno familiar para los usuarios y de fácil navegación.

Palabras clave:

- Android.
- Juez en línea.
- Servicios Web.
- Java.

José A. Flores Roch
Lorenzo A. Todisco López

Abstract

AceptaElReto! is a repository and an online judge for spanish written programming problems that accepts solutions in C, C ++ and Java.

AceptaElReto App! is an Android application that helps mobile (smartphone, tablets) users navigate and interact in a comfortable way with the online judge.

The application will send and receive information through web services. To allow use of the Application offline, we used the cache memory found on the mobile devices to allow users to store information, like problem's solutions.

To facilitate user navigation on our application, we developed our app following the guidelines agreed at the Guide/Webpage of AndroidDevelopers, achieving this way a familiar environment for the users.

Keywords:

- Android.
- Online Judge.
- Web Services.
- Java.

1. Contenido

2. Introducción

2.1. Motivación

Los problemas de programación en primeros años de cursos de ingeniera, en especial en Ingeniería Informática, han desarrollado cierto interés dentro de la comunidad informática, llegándose a crear concursos donde los participantes se enfrentaban a varios de estos problemas proponiendo sus propias soluciones.

Motivados por estos hechos, unos profesores de la facultad de Informática de la UCM decidieron crear un recopilatorio y comprobador de soluciones de este tipo de problemas, de forma que se automatice la corrección de cada problema, además de crear una colección extensa de problemas de programación de primeros años de Ingeniera.

Deciden así montar una aplicación web que cumpla estas funcionalidades y además presentar este trabajo como trabajo de Sistemas Informáticos en su facultad. Apoyados por sus tutores, el proyecto consigue llevarse cabo y presentarse.

Tras este alumbramiento, los tutores del proyecto deciden llevarlo un paso más y modificar la idea de forma que resulte funcional, eficiente y atractiva. Tras lo cual *¡Acepta el reto!* ve la luz.

¡Acepta el reto! (www.aceptaelreto.com) es una aplicación web desarrollada por dos profesores de la Universidad Complutense de Madrid. Se trata de un almacén y *juez en línea* de problemas de programación en español que acepta soluciones en C, C++ y Java.

La rápida asimilación y la alta usabilidad de los dispositivos móviles ha causado una gran demanda de que cualquier contenido web tenga una disposición dedicada a este tipo de dispositivos.

Así, han ido apareciendo en nuestro Google Play Market una infinidad de aplicaciones para nuestros dispositivos Android adaptando aplicaciones Web ya existentes a la comodidad de nuestro móvil o tableta.

Por ello llegamos a la conclusión que una App Android que adaptase al móvil o Tablet la web Aceptaelreto.com podía ser un proyecto interesante y demandado.

2.2. Objetivo

El objetivo principal es trasladar la aplicación web www.aceptaelreto.com a dispositivos Android.

Trasladando así todas o al menos la mayoría de las funcionalidades al Sistema Operativo Android.

Aunque podemos tener nuestras reservas acerca de la comodidad de crear código en un dispositivo de pantalla táctil, decidimos que la consulta de los contenidos ya creados, la interacción entre usuarios por medio de comentarios y el cambio de perfil eran funcionalidades que fácilmente podían ser demandadas por un usuario móvil.

Además llegamos a la conclusión que incluso un usuario motivado podría llegar a crear código en uno de estos dispositivos.

Por lo tanto, decidimos no restar ninguna funcionalidad que tuviera la web para un usuario corriente.

3. Estado del arte

En este apartado, denominado estado del arte, trataremos el estado en el que nos encontramos las herramientas y el entorno en el que vamos a trabajar.

En concreto trataremos dos, en Sistema Operativo (en adelante SO) para el que vamos a crear la aplicación, Android y la aplicación que queremos trasladar y de la cual haremos uso de sus servicios web, AceptaElReto.com.

3.1. Android

Android es un sistema operativo desarrollado por GOOGLE basado en el núcleo Linux diseñado originalmente para dispositivos móviles, tales como teléfonos inteligentes, pero posteriormente se expandió su desarrollo para soportar otros dispositivos tales como Tablet, reproductores MP3, netbook, PC, televisores, lectores de ebook e incluso, se han llegado a ver en microondas y lavadoras.

Lo que lo hace diferente es al estar basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma.

El sistema permite programar aplicaciones en una variación de Java llamada Dalvik. El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda etc.) de una forma muy sencilla en un lenguaje de programación muy conocido y extendido como es java.

Las versiones de Android reciben, en inglés, el nombre de diferentes postres o dulces. En cada versión el postre o dulce elegido empieza por una letra distinta, conforme a un orden alfabético:

- A: Apple Pie (v1.0): Tarta de manzana
- B: Banana Bread (v1.1): Pan de plátano
- C: Cupcake (v1.5): Panque
- D: Donut (v1.6): Rosquilla
- E: Éclair (v2.0/v2.1): Pepito
- F: Froyo (v2.2): Yogurt helado
- G: Gingerbread (v2.3): Pan de jengibre
- H: Honeycomb (v3.0/v3.1/v3.2): Panal de miel
- I: Ice Cream Sandwich (v4.0): Sandwich de helado
- J: Jelly Bean (v4.1/v4.2/v4.3): Gominola
- K: KitKat (v4.4): Kit Kat
- L: Lollipop (v5.0/v5.1): Piruleta
- M: Marshmallow (v6.0): Malvavisco

Llegando a un nivel de API del 22.

3.2. Servicios Web

En el mundo actual, Internet es la puerta grande de entrada a la información, y a un conjunto de servicios ofrecidos por empresas o personas independientes creando de esta forma una interacción *Servicio Online – Persona*. Aun así muchas veces esta interacción y el proceso de la misma se deben automatizar, por lo que esta interacción *Servicio Online – Persona* pasa a ser *Servicio Online – Aplicación*. Para favorecer este tipo de interacciones se han creado un conjunto de estándares que se ha denominado Servicios Web.

La World Wide Web Consortium lo define como "…un sistema de software diseñado para soportar interacción interoperable máquina a máquina sobre una red. Este tiene una interface descrita en un formato procesable por una máquina (específicamente WSDL). Otros sistemas interactúan con el servicios web en una manera prescrita por su descripción usando mensajes SOAP, típicamente enviados usando HTTP con una

serialización XML en relación con otros estándares relacionados con la web" [1]. Se puede definir de manera más sencilla como un conjunto de tecnologías estándares de Software para el intercambio de datos entre aplicaciones tales como SOAP, WDSL y UDDI. Estos pueden ser desarrollados en una gran variedad de lenguajes para ser implementados sobre muchos tipos de redes de computadores. El éxito de la Interoperabilidad se consigue gracias a la adopción de protocolos y estándares abiertos. The Organization for the Advancement of Structured Information Standards y el World Wide Web Consortium son los responsables de la estandarización y arquitectura de los servicios web.

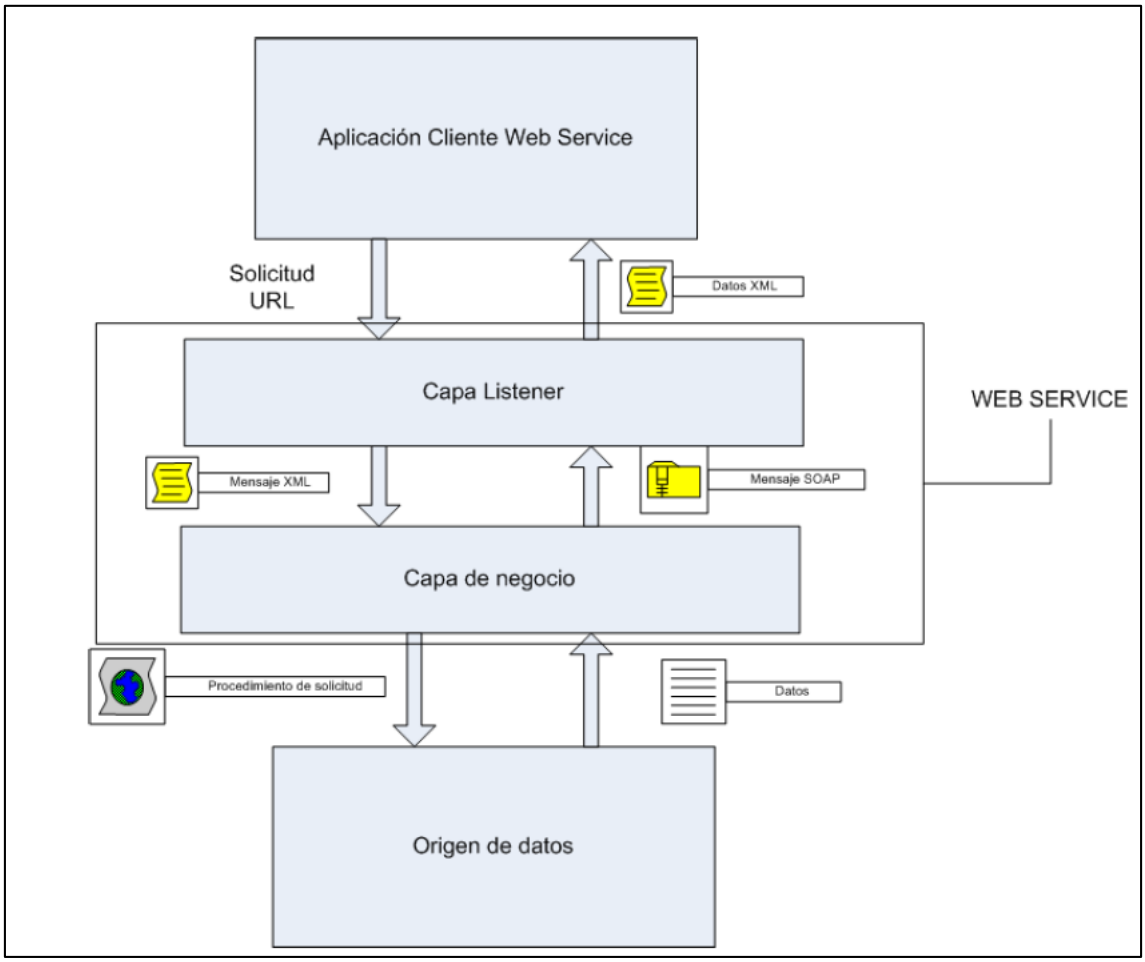


Figura 1. Funcionamiento de un Servicio Web

3.3. AceptaElReto.com

¡Acepta el reto! es un almacén y *juez en línea* de problemas de programación en español que acepta soluciones en C, C++ y Java.

No es un mero listado de problemas, sino mucho más, es un corrector automático.

¡Acepta el reto! nace como una iniciativa de dos profesores de la Facultad de Informática de la Universidad Complutense de Madrid. En 2011 habían contribuido a poner en marcha ProgramaMe, el concurso de programación para alumnos de Ciclos Formativos de Formación Profesional. Además, habían desarrollado varios problemas similares a los de los concursos para realizar pruebas de evaluación continua a sus alumnos de grado.

Todos los problemas creados dormían plácidamente en un cajón y no podían ser aprovechados por estudiantes posteriores más allá de resolverlos sin poder probar su corrección. Debido a eso, decidieron poner en marcha el desarrollo de un juez on-line similar a otros existentes (el más conocido el de la Universidad de Valladolid) pero centrado en problemas en español para alumnos de Ciclos Formativos y primeros años de Universidad.

En el curso académico 2013-14 se ponen manos a la obra con la implementación del *backend*, y delegan el *frontend* web a estudiantes que desarrollan dos versiones preliminares del sitio. Ambas sirven como prueba de concepto y permiten poner a prueba la infraestructura. En los últimos meses de 2013, sustituyen esas pruebas de concepto por una implementación nueva del *frontend*, y en febrero de 2014 *¡Acepta el reto!* ve finalmente la luz.

El equipo de Acepta El Reto está compuesto por los profesores Marco Antonio Gómez Martín y Pedro Pablo Gómez Martín (ambos tutores de este proyecto).

Y otros antiguos miembros, Luis Hernández Yañez (Tutor de este proyecto), Jéssica Martín Jabón, Javier Martín, Moreno-Manzanaro, Pablo Suárez Díaz, Luis María Costero Valero, Jesús Javier Domenech Arellano.

Esta aplicación web nos proporcionara todos los servicios Web necesarios para poder trasladar Acepta El Reto al SO Android.

La Web actual se muestra de la siguiente manera:

Figura 2. Inicio Aceptaelreto.com

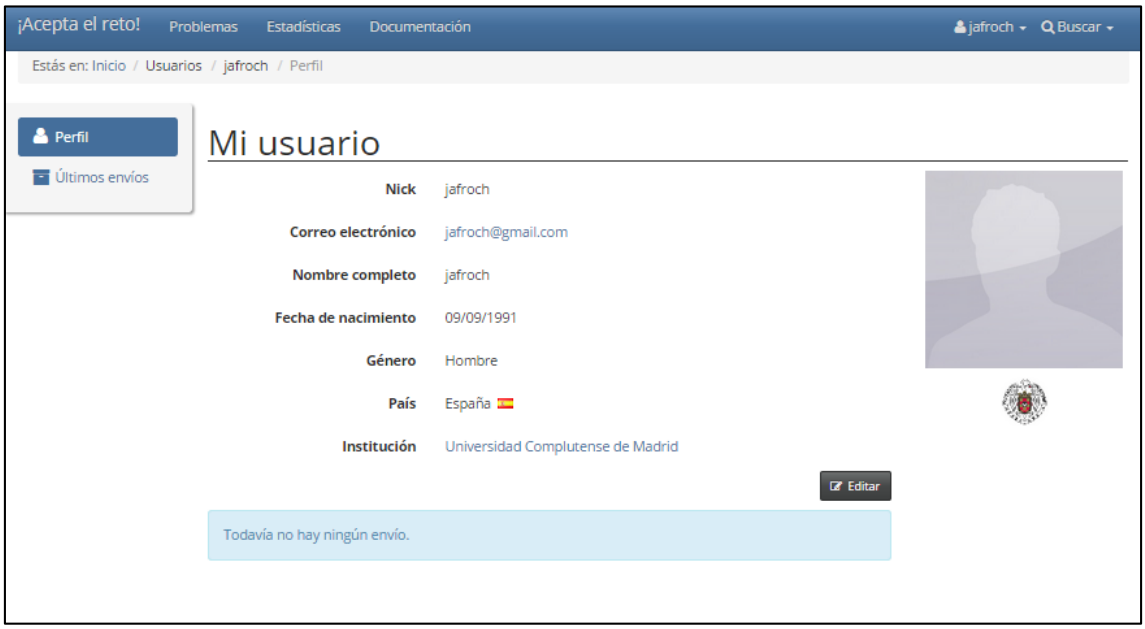


Figura 3. Perfil Aceptaelreto.com

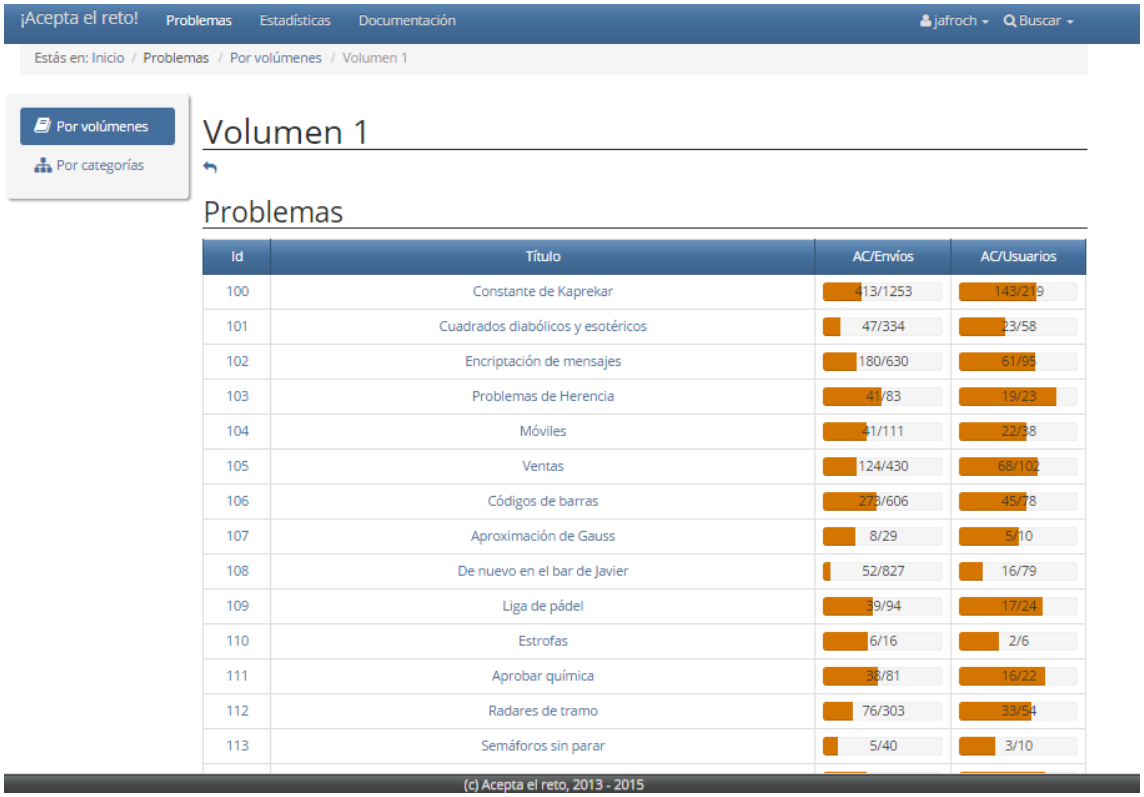


Figura 4. Problemas Aceptaelreto.com

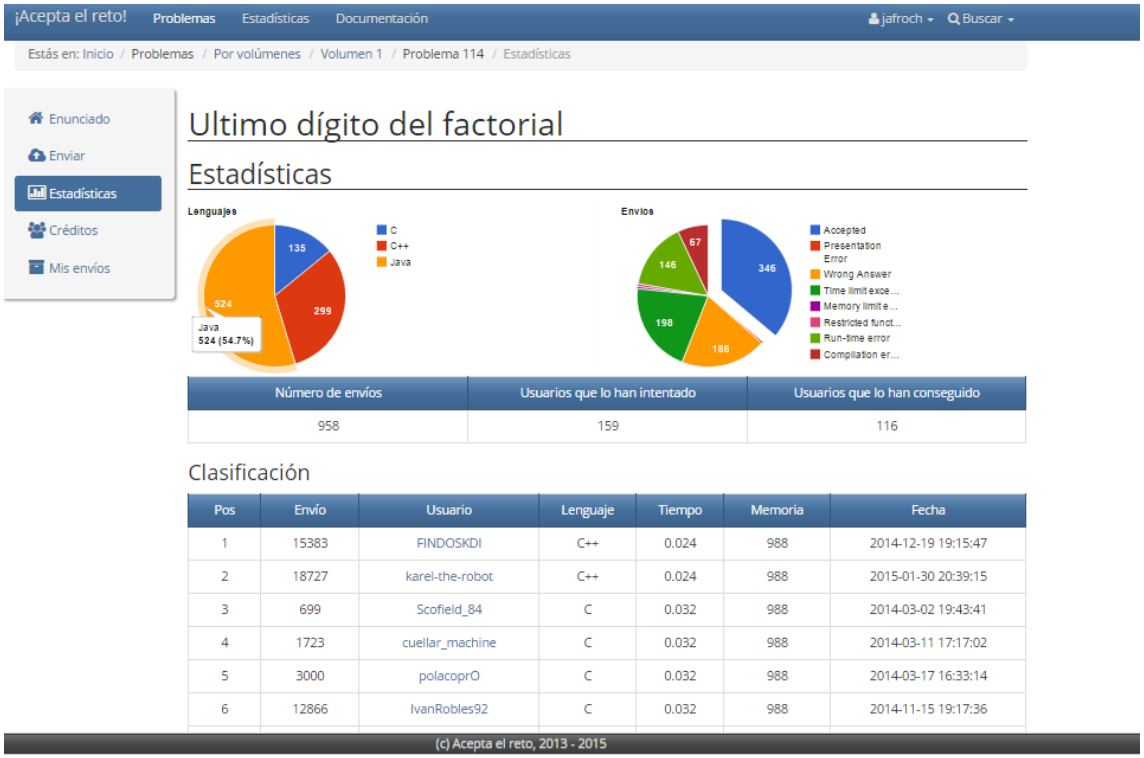


Figura 5. Estadísticas Aceptaelreto.com

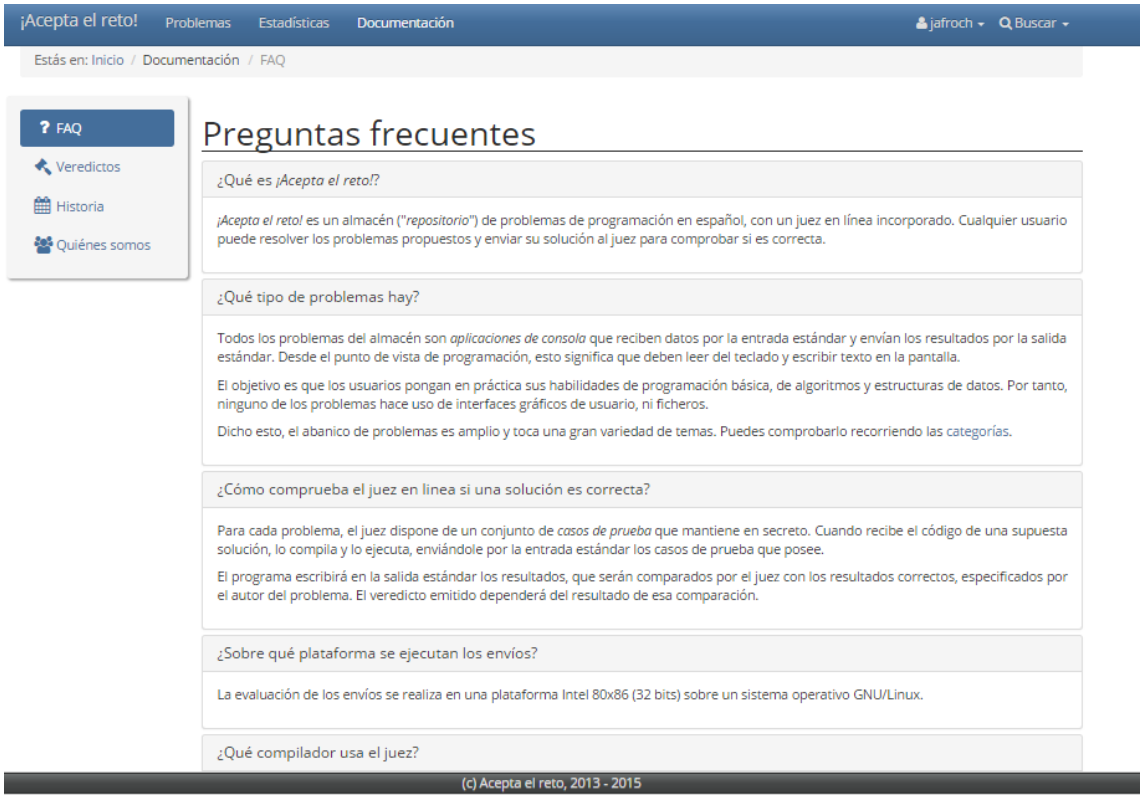


Figura 6. Faq Aceptaelreto.com

4. Fase de Diseño

En esta fase realizamos una propuesta de la como debe actuar e interactuar la aplicación con el usuario a nivel de diseño, que conseguirá darnos una primera imagen de cómo vamos a plantear el diseño de nuestra aplicación.

La estrategia vamos a centrarla en la creación de una aplicación móvil que sea capaz de ofrecer todas las funcionalidades que ofrece la aplicación web a un usuario de Acepta El Reto priorizando una solución que ofrezca un uso intuitivo y sencillo.

4.1. Vista conceptual

La arquitectura para el diseño de la aplicación debe estar cimentada en los requisitos funcionales y no funcionales de la aplicación, es decir la arquitectura debe diseñarse para lograr el objetivo final, conseguir que la aplicación Móvil compita en funcionalidad con la aplicación Web.

Así, un diagrama conceptual teniendo en cuenta la vista que el usuario tendrá de la aplicación será como la mostrada en la figura 3.

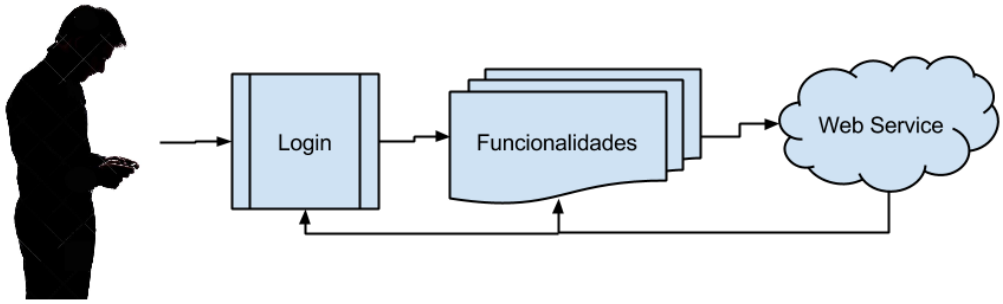


Figura 7. Vista conceptual

4.2. Arquitectura Modelo-Vista-Controlador

La arquitectura Modelo-Vista-Controlador (MVC) es un patrón de diseño que separa la lógica de negocio, la interfaz de usuario y los datos con los que se trabaja. Divide la aplicación en tres partes:

- Un modelo que contiene la información con la que se trabaja, es decir, los datos.
- Vistas que gestionan cómo se muestra la información al usuario de una manera accesible.
- Uno o varios controladores que realizan las funciones correspondientes en el modelo como respuesta a eventos procedentes generalmente de la vista.

Una ventaja de usar el MVC es que se incrementa el nivel de reutilización de los diferentes módulos software y de la aplicación en general. Para ello el modelo no puede tener acceso a ninguna clase del Controlador ni de la Vista. Además, el cambio de modelo no debe afectar a las vistas, y el controlador debe ver las clases del modelo, pero

no de la vista. En la figura 4 pueden observarse la manera de interactuar entre los distintos componentes que forman parte de la arquitectura MVC.

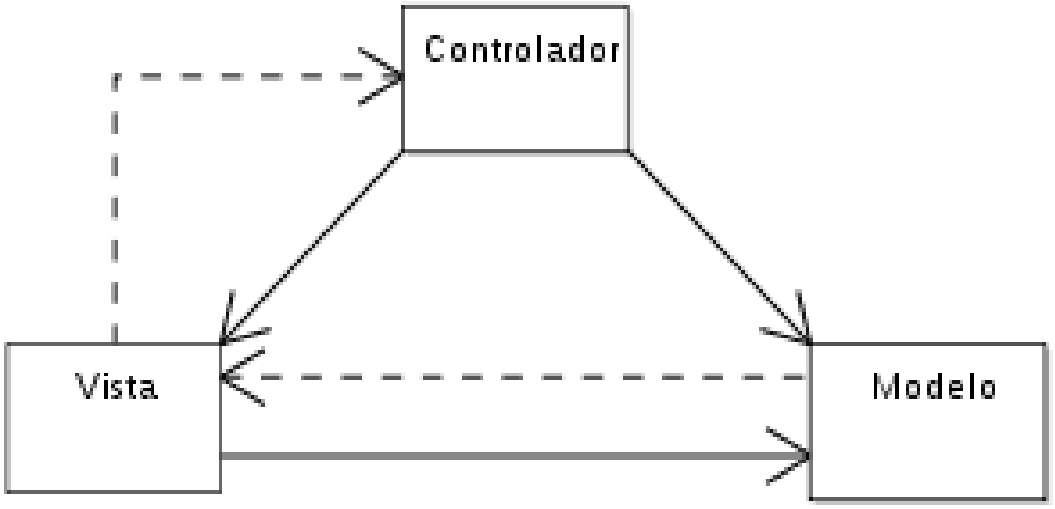


Figura 8. Diagrama modelo-vista-Controlador

El uso del patrón MVC permite la evolución de forma separada de cada una de las partes, lo que proporciona una mayor flexibilidad a la hora de desarrollar los componentes del sistema.

Aunque existen diferentes implementaciones de MVC, normalmente el flujo que sigue es el siguiente:

- Usuario interactúa con la interfaz de usuario.
- El controlador gestiona el evento de entrada.
- El controlador informa al modelo sobre la acción elegida por el usuario, pudiendo acceder al modelo y modificarlo si fuera necesario de acuerdo a dicha acción.
- Se modifica la vista, tomando los datos del modelo.
- La interfaz de usuario espera nuevas interacciones del usuario.

Las ventajas sobre el uso de MVC son las que expresan a continuación:

- Teniendo un mismo modelo, es posible conservar diferentes vistas.
- Es posible añadir nuevas vistas sin la necesidad de modificar el modelo.
- Menor acoplamiento entre vistas y modelo.

- Diseño más claro.

Evidentemente, este modelo no es único, existen muchas variaciones e interpretaciones, si bien, lo importante es separar los tres elementos de que consta, algo que cumple el utilizado en este trabajo.

4.3. Diagrama de clases

El diagrama de clases es el diagrama principal para el diseño estático, presentando y describiendo las clases y objetos del sistema con sus relaciones estructurales y de herencia. En la definición de clases se muestran los atributos y métodos incluidos en la clase correspondiente.

En la siguiente figura podemos ver los 3 paquetes de clases en que hemos dividido nuestro proyecto.

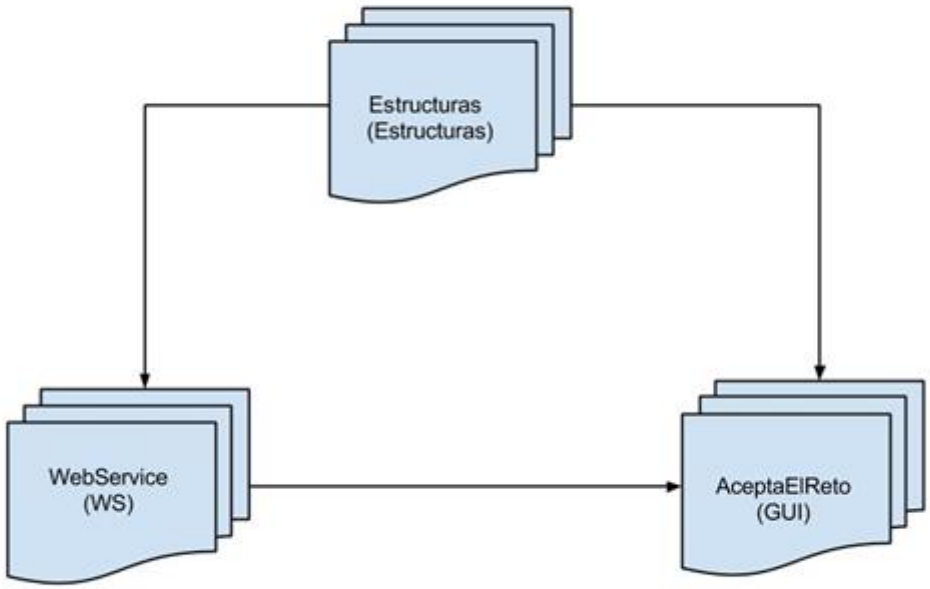


Figura 9. Diagrama paquetes

Estructuras: Contiene todos los tipos de las variables java que usamos para referirnos a los datos que intercambiamos con el Servidor web, la GUI hace todo tipo de manejo de estas clases mientras que el WebService simplemente las usa para dar formato a las consultas y pasar informacion pedida a la GUI.

WebService: Contiene todas la clases necesarias para que la GUI se abstraiga de la conexión con el ServicioWeb. Además ofrece clases de encapsulamiento de consulta, de forma que con unos simples métodos el programador de la GUI puede ser capaz de hacer cualquier consulta al Servidor y obtener un objeto que conoce.

GUI: (Interfaz gráfica) Contiene todo lo referente a la parte visual de la aplicación, incluyendo las distintas funcionalidades para el usuario y las distantas ventanas que puede ver. Siendo una aplicación Android, la accesibilidad y la estética cuentan mucho a la hora de valorar la aplicación. Este paquete esta formado por dos partes, 1) la interfaz gráfica, que contiene todos los Fragmentos y Actividades pertinentes a la interfaz, y 2) "Tools", que contiene aquellas clases usadas como herramientas (adaptadores, gráficos, manejo de imagenes) por la interfaz.

Si entramos mas en detalle, el diagrama más correcto es el de la Figura 10, donde vemos que las clases que antes hemos llamado clases de encapsulamiento de consulta, son la clase **Traductor** y **CallerWS**, que se encargan de hacer de herramientas de acceso a la informacion que ofrece el Servidor.

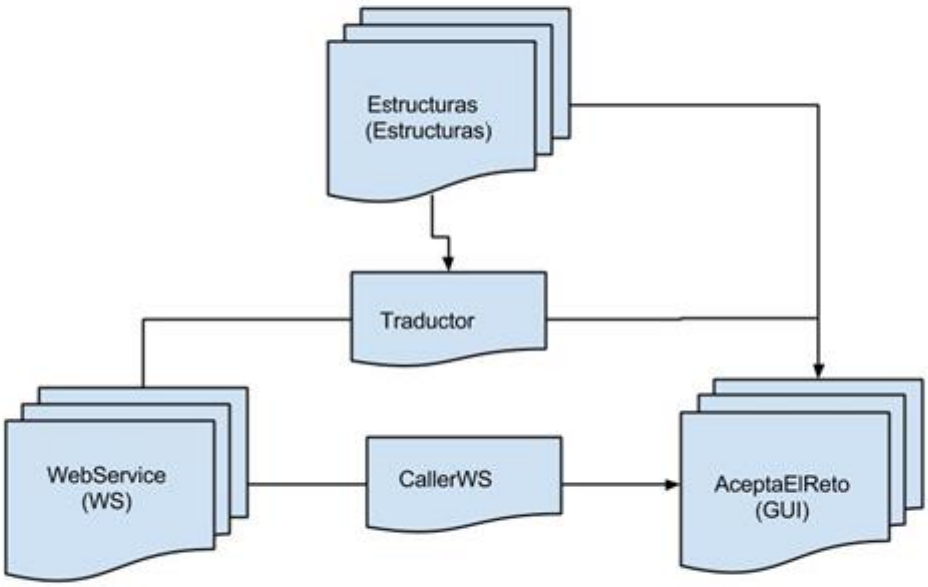


Figura 10. Relacion entre paquetes

A continuación mostramos los diagramas UML de cada paquete para que podamos ver en un mayor detalle lo que contiene cada paquete y como se relacionan las clases entre ellas.

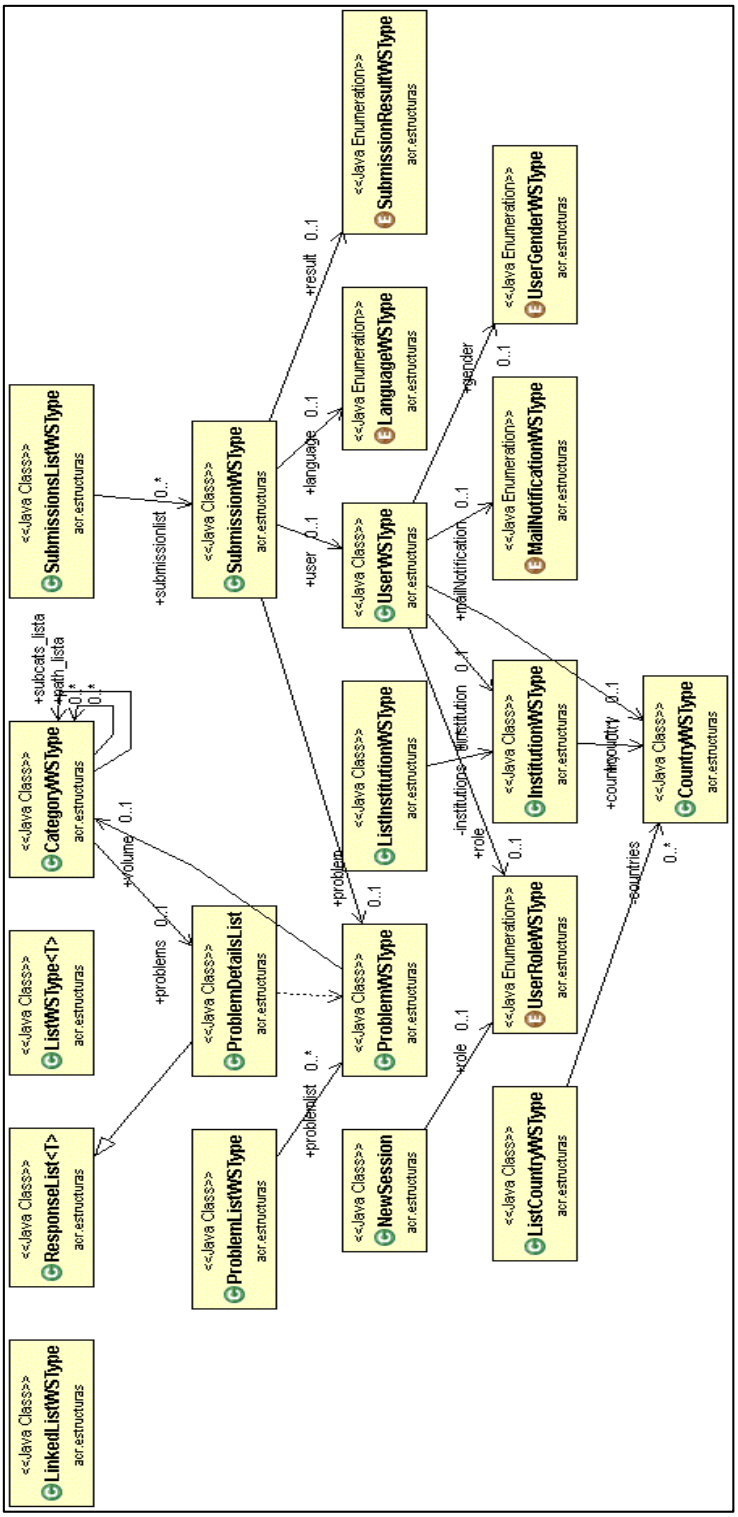


Figura 11. Diagrama UML Estructuras

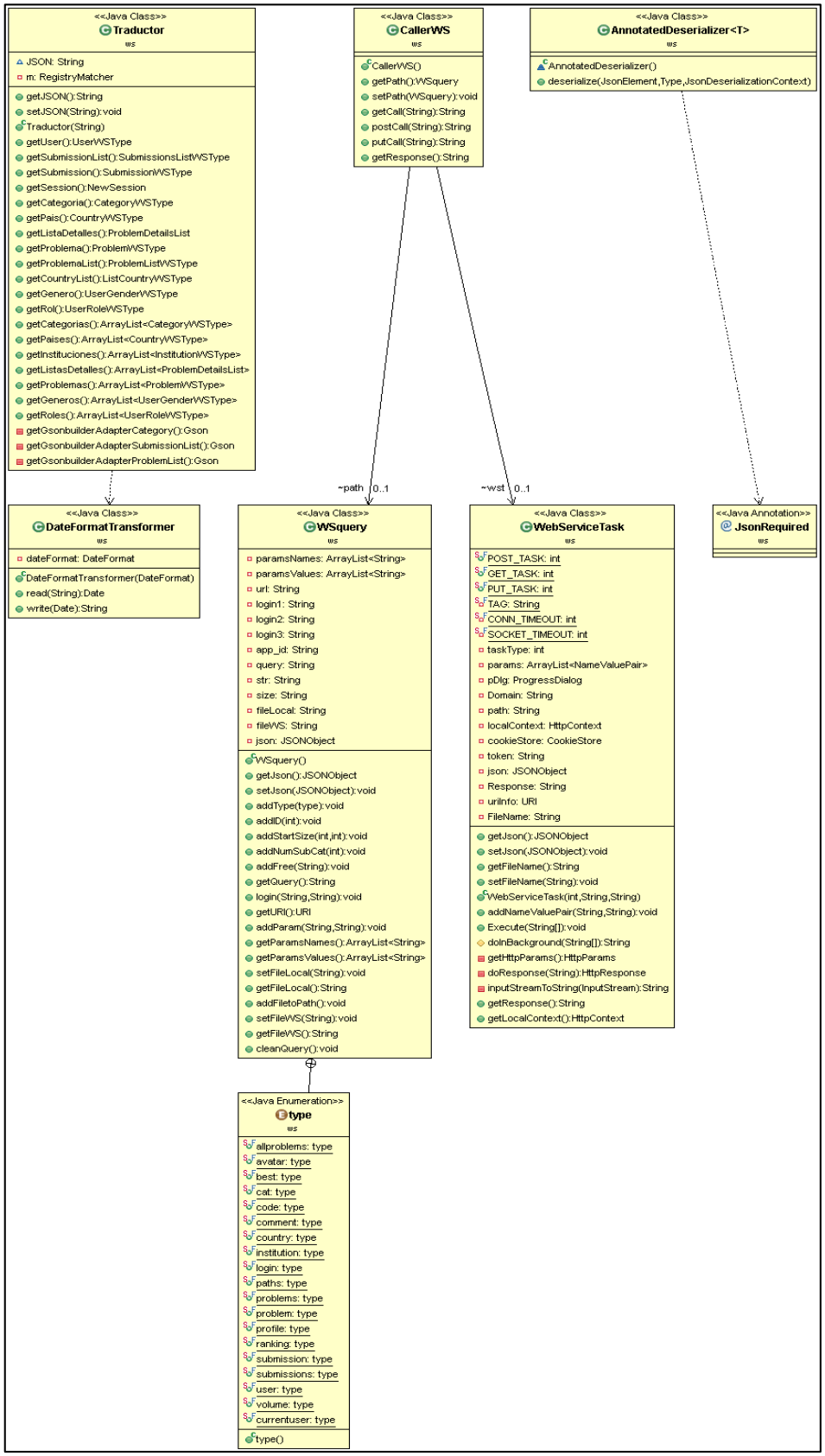


Figura 12. Diagrama UML WS

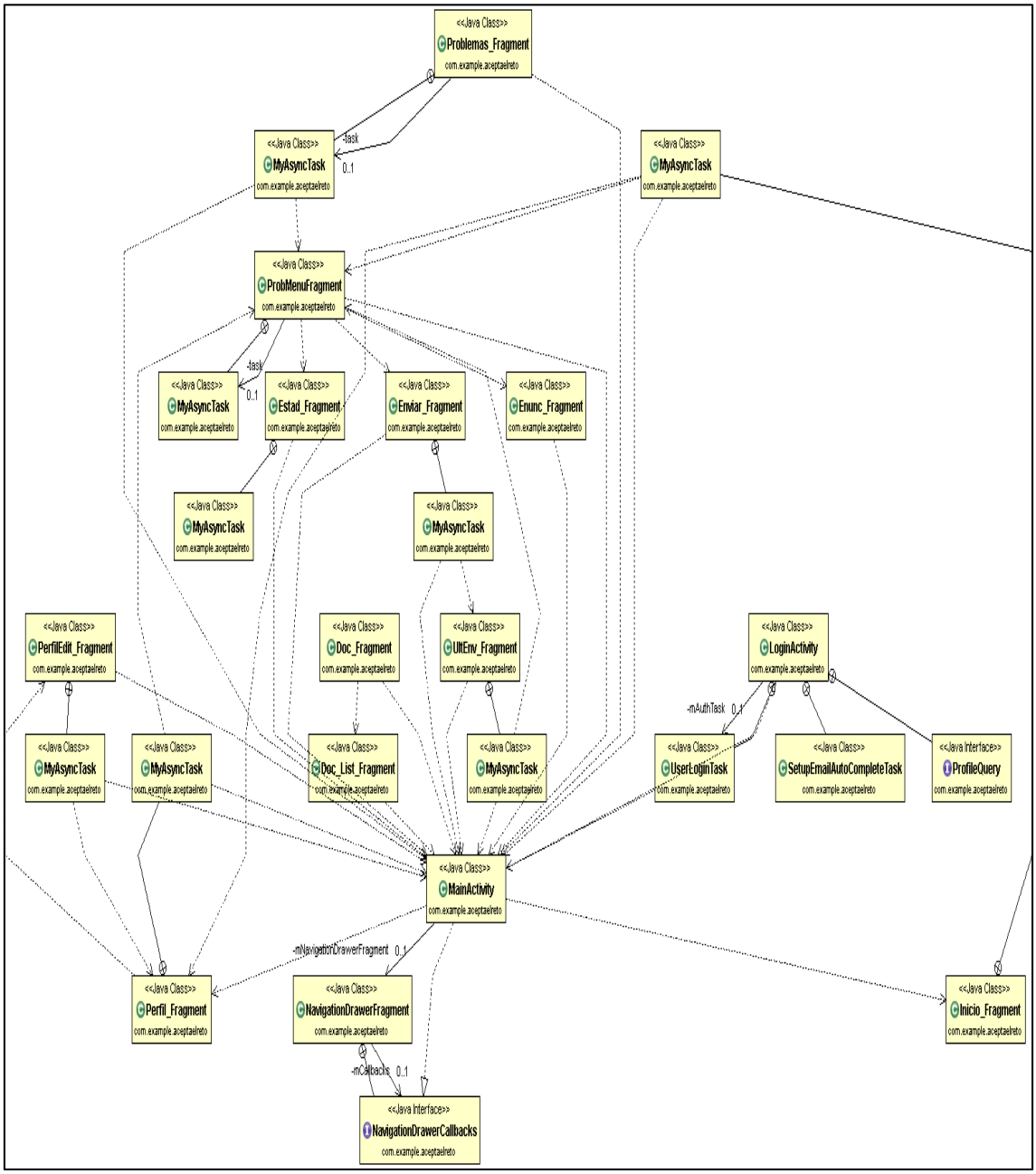


Figura 13. Diagrama UML GUI

5. Fase de implementación

5.1. Descripción

En esta fase veremos con detalle la implementación que se ha llevado a cabo para obtener el proyecto del que hemos estado hablando, AceptaElRetoApp.

Se verán las distintas partes en las hemos dividido el proyecto, modularizandolo de esta manera. Y entraremos a describir los puntos más importantes y llamativos.

Todo este proceso de implementación se adecua a todo lo desarrollado en la fase de análisis y en la fase de diseño.

Lenguaje de programación: JAVA

Al ser el lenguaje en el que Android permite la programación nos hizo que no hubiera elección, aunque esto nos supuso más bien una ventaja más que un inconveniente ya que probablemente sea el lenguaje en el que recogemos más experiencia.

Algunas de las ventajas de Java son:

- Orientado a objetos

  Se facilita mucho el uso de técnicas como la modularidad, herencia, abstracción, polimorfismo y encapsulamiento, las cuales se resultan muy útiles para los objetivos, sobre todo de cara a una futura evolución y ampliación de la aplicación.

- Open Source

  Gran flexibilidad y que sea capaz de evolucionar según las necesidades de mercado y usuarios constituyen una gran ventaja.

- Independiente de la plataforma

  El hecho de que la aplicación sea ejecutable independientemente del sistema operativo o la arquitectura de la máquina. Aunque en nuestro caso esto estaba limitado al tratarse de un proyecto para Android.

- Recolector de basura

  La generación de basura está muy ligada a la orientación a objetos. En java, es el propio lenguaje el que se encarga de destruir automáticamente los objetos que se han quedado sin referencia, liberando así la memoria. Además, en nuestro caso particular, tratamos con un gran número de objetos así que esta característica se vuelve esencial.

- Facilidad de aprendizaje

  Java es un lenguaje muy sencillo de aprender y muy manejable para enfrentarse a nuevos retos. Su gran comunidad de programadores a nivel mundial permite ampliar rápidamente conocimientos en el caso de ser necesario ya que hacen que resulte fácil encontrar información y ayuda.

- Entornos de desarrollo

  Al ser un lenguaje tan utilizado posee muchas alternativas a la hora de elegir con qué herramienta queremos trabajar. Aunque para la programación en Android ese número se ve algo más reducido sigue siendo considerable. En nuestro caso hemos elegido Eclipse por su capacidad para añadir módulos de diseño que nos han ayudado en varios de los procesos de creación, Interfaz, pruebas en máquinas virtuales, actualización de APIs etc. Además resaltar que ha sido un entorno en el que hemos trabajado con anterioridad y nos sentíamos cómodo con él.

- Librerías

  Existe una gran variedad de librerías que facilitan el tratamiento de operaciones y por tanto el trabajo de desarrollo. El poder trabajar con estas librerías facilita enormemente el trabajo de implementación. Aun así no todas las librerías de Java son admisibles por Android, lo que nos complicó el trabajo en más de una ocasión obligándonos a buscar otras librerías similares pero menos conocidas, el caso que más nos perjudico fue la librería *"javax.xml.bind"*, que no al no ser admitida por el entorno Android tuvimos que sustituir por la librería "Simple".

5.2. Librerías

Como hemos hablado antes, Java ofrece una gran cantidad de librerías que añaden funcionalidades, entre ellas, algunas ofrecen soporte para Android. Las librerías que

hemos usado y que pasamos a detallar a continuación nos han proporcionado una gran ayuda y han sido de mucha utilidad en nuestro proyecto.

5.2.1. *Jackson*

Es una librería de utilidad de Java que nos simplifica el trabajo de serializar (convertir un objeto Java en una cadena de texto con su representación JSON), y deserializar (convertir una cadena de texto con una representación de JSON de un objeto en un objeto real de Java) objetos JSON.

En nuestro proyecto es el encargado de toda la serialización y deserialización de los las respuestas en JSON recibidas del servicio web.

```
public UserWSType getUser() throws Exception{
        if(this.JSON.startsWith("<?xml")){
                Serializer serial = new Persister(m);
                UserWSType data = serial.read(UserWSType.class, this.JSON);
        return data;
        }else{
                Gson gson = new GsonBuilder().create();
                JsonParser parser = new JsonParser();
                JsonObject data = parser.parse(this.JSON).getAsJsonObject();
                return gson.fromJson(data, UserWSType.class);
        }
```

5.2.2. *Gson*

Librería JAVA de código abierto implementada por GOOGLE que permite la serialización y deserialización entre objetos Java y sus representaciones en JSON/XML.

En nuestro proyecto esta librería es la encargada de transformar los elementos XML que obtenemos del servicio web.

Aunque intentamos que primase el uso de objetos JSON, en ocasiones obtenemos codificaciones en XML, lo que nos hizo tener que plantear los dos casos: que la información viniera en formato JSON o en XML, esto nos obligaba a tratar los dos casos para todo tipo de objeto y de esta forma además abstragimos la clase traductor, que se encarga de la serialización y deserialización, del resto de web Service.

```
public UserWSType getUser() throws Exception{
        if(this.JSON.startsWith("<?xml")){
                Serializer serial = new Persister(m);
                UserWSType data = serial.read(UserWSType.class, this.JSON);
        return data;
        }else{
                Gson gson = new GsonBuilder().create();
                JsonParser parser = new JsonParser();
                JsonObject data = parser.parse(this.JSON).getAsJsonObject();
                return gson.fromJson(data, UserWSType.class);
        }
```

Figura 14. Uso de Gson en ¡AceptaElRetoApp!

5.2.3. *Simple XML*

Simple es una Librería de alto rendimiento para JAVA que permite la serialización y configuración del marco XML en objetos JAVA. Su objetivo es proporcionar un marco XML que permite el desarrollo rápido de configuración XML y sistemas de comunicación. Este marco ayuda al desarrollo de sistemas de XML con el mínimo esfuerzo y errores reducidos. Ofrece la serialización y deserialización de objetos completos, manteniendo cada referencia encontrada.

El uso de esta librería surgió como necesidad a encontrar una librería alternativa a *"javax.xml.bind"* que nos permitiera añadir etiquetas XML en nuestras clases, puesto que esta última no permitía su uso dentro de Android.

```
@Root(name="country")
//@XmlType(propOrder={"code", "nombre", "name"})
public class CountryWSType {

        /** Código ISO */
        @Element(name="code")
        public String code;

        /** Nombre en español */
        @Element(name="nombre")
        public String nombre;

        /** Nombre en inglés */
        @Element(name="name")
        public String name;

        /** Constructor sin parámetros. No inicializa nada...
         * Está para que funcione el marshall/unmarshall.
         */
        public CountryWSType() {}
}
```

Figura 15. Uso de Simple en ¡AceptaElRetoApp!

Como vemos en las imágenes, se añaden unas etiquetas con el marcador "@" de forma que al realizar el marshalling and unmarshalling se identifiquen correctamente las diferentes clases y los atributos que la constituyen.

5.2.4. *Volley*

Volley es una librería desarrollada por google para el manejo de imágenes y peticiones http desde aplicaciones Android. Todas las llamadas realizadas a través de Volley son de tipo asíncronas, es decir, todas las llamadas se realizan desde el hilo principal, eliminando la necesidad de crear Asyntasks y facilitando el networking y carga de imágenes desde el servidor.

Dentro de las librerías que ofrecen esta funcionalidad nos decidimos por ésta por ser la que nos permitía realizar las tareas en menos pasos, de foma más sencilla y sin necesidad de crear clases adicionales.

5.2.5. *Picasso*

Picasso, como Volley, es una librería que se encarga de la descarga y almacenamiento de imágenes en la caché del dispositivo, para obtener así un mejor funcionamiento y tratamiento de las mismas.

Hemos utilizado esta librería para el tratamiento de imágenes de gran tamaño, como los enunciados de los problemas, ya que permite acondicionar las imágenes a los tamaños standard admitidos por nuestra aplicación.

5.3. Herramientas de Trabajo

Aquí describimos las diferentes herramientas que nos han hecho posible y que nos han ayudado a terminar con éxito nuestra aplicación móvil.

5.3.1. *Eclipse*

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Como ya hemos comentado, el entorno de programación que habíamos elegido por nuestra previa experiencia ha sido eclipse.

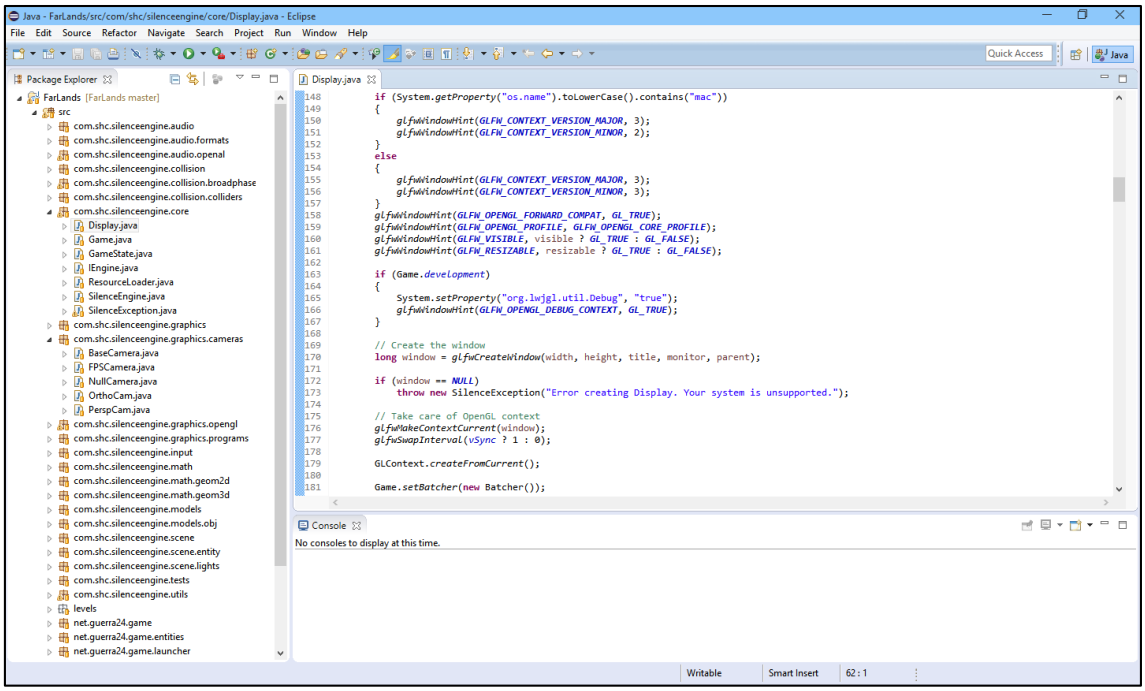


Figura 16. Captura de Eclipse

Eclipse nos ha facilitado un entorno familiar, cómodo y sobre todo completo.

Además para facilitarnos el trabajo incluimos en nuestra configuración de eclipse varios plug-in auxiliares para añadir funcionalidades que nos facilitaran la tarea, como es el

caso de crear diagramas UML, una funcionalidad que nos permitía gestionar nuestro proyecto en Github, otra que nos permitía editar visualmente Interfaces Graficas para Android, etc. Etc.

5.3.2. *GitHub*

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por *GitHub, Inc.* (anteriormente conocida como *Logical Awesome*). Desde enero de 2010, GitHub opera bajo el nombre de *GitHub, Inc.* El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

Figura . Logo GitHub

GitHub nos ha sido de mucha utilidad, nos ha permitido trabajar de manera colaborativa entre nosotros cada uno desde su puesto y sin necesidad de andar con complicados transportes y pegados de código que solo hubieran añadido complicaciones. Además gracias al software que facilita eclipse importar y gestionar un proyecto en eclipse usando GitHub ahorra secuencias de comandos complicadas en consola.

5.3.3. *AVD*

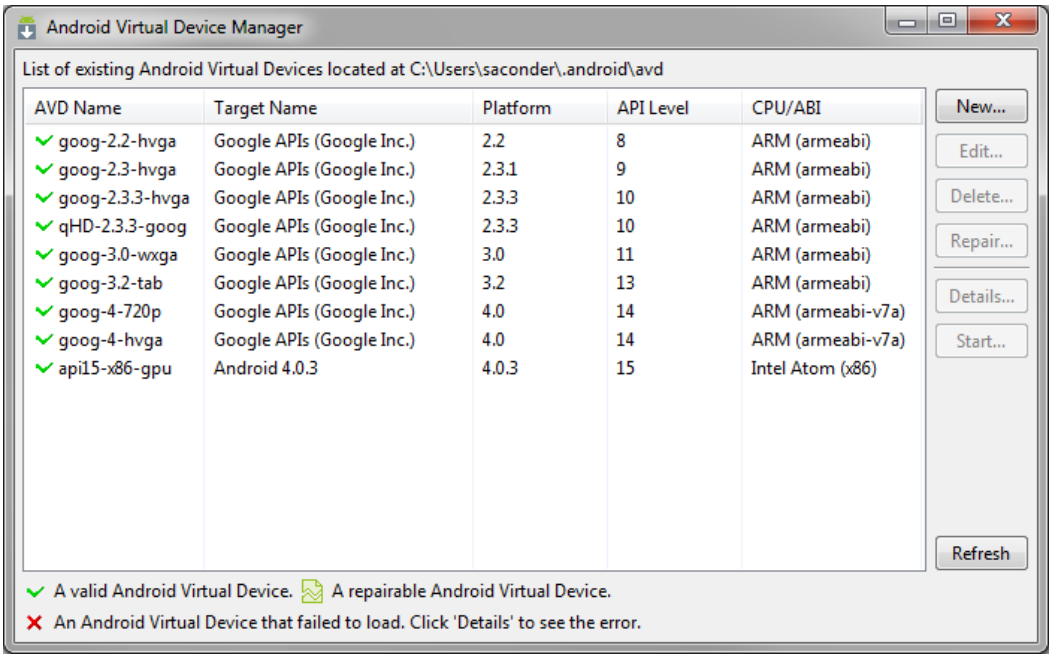


Figura 17. Android Virtual Device Manager

Android Virtual Device, se trata de un dispositivo virtual de Android.

También contaba con un plug-in en Eclipse que nos permitía gestionar todas las máquinas virtuales y enviar pruebas directamente desde nuestro entorno de programación.

Figura 18. Captura de ADV arrancado

AVD nos ha permitido probar nuestra aplicación, descubrir errores de ejecución en Android en tiempo real, probar funcionalidades y visualizar la interfaz de primera mano, además también permite configurar cualquier dispositivo Android de forma que puedas probar tu app en cualquier momento en tu dispositivo real.

5.4. Detalles de Implementación

En este apartado explicaremos con más detalle el código. lo hemos dividido en tres apartados que corresponden a los tres paquetes en los que hemos dividido el código, hemos intentado que estos tres paquetes consigan abstraerse unos de otros, de forma que la sustitución de cualquiera de ellos por otro que ofrezca las mismas funcionalidades sea lo más sencillo posible y que no afecte a ninguna de las otras partes.

Destacar de la implementación que cada persona del grupo ha desarrollado una de ellas por su parte (interfaz y Web Service) de forma que ofrecer una manera sencilla de hacer uso del paquete ajeno era una necesidad primordial, puesto que facilitaba y simplificaba el trabajo reduciendo las explicaciones de detalles de implementación al mínimo.

5.4.1. *Estructuras*

En este paquete hemos especificado todas las clases que definían las estructuras de datos que intercambian el WS y la Interfaz.

Las clases de este paquete, al ser representaciones de datos de los objetos que devuelve el Servicio Web en forma de JSON o XML, debemos tenerlas serializados, para que sea posible hacer MARSHALING y UNMARSHALING (Acción de la que se encargaran la librería Simple y Jackson).

Ejemplo de clase serializada para XML:

```
@Root(name="user")
//@XmlType(propOrder={"id", "nick", "email", "name", "gender", "country", "institution", "mailNotification", "role", "birthday",
public class UserWSType {

	// Identificador del usuario.
	@Element(name="id")
	public Integer id;

	// Nick utilizado para hacer login.
	@Element(name="nick")
	public String nick;

	// Dirección de correo electrónico
	@Element(name="email", required = false)
	public String email;

	// Nombre completo
	@Element(name="name",required = false)
	public String name;

	// Género
	@Element(name="gender",required = false)
	public UserGenderWSType gender;

	// País
	@Element(name="country",required = false)
	public CountryWSType country;

	// Institución a la que pertenece
	@Element(name="institution",required = false)
	public InstitutionWSType institution;

	// Tipo de notificación por mail utilizada
	@Element(name="mailNotification",required = false)
	public MailNotificationWSType mailNotification;

	// Role del usuario
	@Element(name="role",required = false)
	public UserRoleWSType role;

	// Fecha de nacimiento
	@Element(name="birthday",required = false)
	public Date birthday;
```

Figura 19. Clase UserWSType serializada.

Para serializarla, usamos las etiquetas “@....” que nos facilita la librería Simple.

5.4.2. *Interfaz*

Aquí especificamos la parte visual y funcional de la aplicación, es la encargada de ofrecer una vista atractiva de la aplicación así como ejecutar todas las funcionalidades que ofrece, llamando al Web Service para obtener lo que necesite del Servidor.

En este paquete encontraremos dos Actividades, la actividad de **Login**, donde el usuario accede a la aplicación y donde recogemos el cookie de identificación de sesión una vez el usuario se ha conectado correctamente en la aplicación; y la actividad **Main**, es la responsable del manejo y navegación de la aplicación, en ella encontraremos la barra de navegación, así como los fragmentos que componen nuestra aplicación.

```
public class MainActivity extends ActionBarActivity implements
        NavigationDrawerFragment.NavigationDrawerCallbacks {

    private NavigationDrawerFragment mNavigationDrawerFragment;

    //ATRIBUTOS PARA MANEJO DEL MENU.
     CharSequence mTitle;
     String[] opcionesMenu;
     DrawerLayout drawerLayout;
     ListView drawerList;
     GridView tablaPerfil;
     public static String Token;
     public static String myId;
     public static int numTransaction;
     public static boolean probList;
     Bundle args;

    protected void onCreate(Bundle savedInstanceState) {

    @Override
    public void onNavigationDrawerItemSelected(int position) {
          // update the main content by replacing fragments

          FragmentManager fragmentManager = getSupportFragmentManager();
          fragmentManager.popBackStack(null, FragmentManager.POP_BACK_STACK_INCLUSIVE);
          numTransaction = 0;
          probList = false;
          switch (position+1) {
            case 1:
              fragmentManager.beginTransaction().replace(R.id.container,
              Inicio_Fragment.newInstance(position + 1,Token)).addToBackStack(null).commit();
              break;
            case 2:
              fragmentManager.beginTransaction().replace(R.id.container,
              Perfil_Fragment.newInstance(position + 1,Token,0)).addToBackStack(null).commit();
              break;
```

Figura 20. Clase Main

Para cada "tipo" de ventana que veremos en nuestra App, hemos realizado un Fragmento encargado de producir la visualización requerida y lo cual permite versatilidad y optimización de diseño. El uso de Fragmentos a la hora de crear nuestra interfaz, nos permitió la reutilización de los mismos para diferentes ventanas, evitando así generar código repetido y manteniendo una organización limpia de nuestro proyecto.

```
/*
 * clase que genera el fragment de Inicio
 */
public class Inicio_Fragment extends Fragment {
    private static final String ARG_SECTION_NUMBER = "section_number";

    private CallerWS ws;
    private WSquery path;
    private boolean p;
    private Spinner sp;
    private Button buscar;
    private EditText txtbuscar;
    private String tk;
    private ExpandableListAdapter listAdapter;
    private ExpandableListView expListView;
    private String[] groups;
    private String[][] children;
    Bundle token;

    public static Inicio_Fragment newInstance(int sectionNumber, String tk) {
        Inicio_Fragment fragment = new Inicio_Fragment();
        Bundle args = new Bundle();
        args.putInt(ARG_SECTION_NUMBER, sectionNumber);
        args.putString("TOKEN", tk);
        fragment.setArguments(args);
        return fragment;
    }

    public Inicio_Fragment() {

    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {


        View rootView = inflater.inflate(R.layout.inicio_layout, container, false);
```

Figura 21. Ejemplo de Fragmento

A continuación detallaremos los Fragmentos utilizados y sus funcionalidades:

- NavigationDrawerFragment: Fragmento encargado de la funcionalidad de la barra de navegación.
- Inicio_Fragment: Ventana de bienvenida a la App y buscador de problemas/usuarios.
- Perfil_Fragment: Información de usuario.
- PerfilEdit_Fragment: Edición de información de usuario.
- UltEnv_Fragment: Tabla que proporciona la información referente a los últimos envíos tanto del usuario como los de la comunidad de AceptaElReto.
- Problemas_Fragment: Lista de exploración de problemas por Categorías o Volúmenes.

- ProbMenu_Fragment: Menú de exploración de un problema en específico (Enunciado, Enviar, Estadísticas).
- Enunc_Fragmente: Imagen del enunciado del problema.
- Enviar_Fragment: Editor de solución de problemas.
- Estad_Fragment: Estadísticas generales de un problema.
- Doc_Fragment: Meú de exploración de la documentación de AceptaElRetoApp.
- Doc_List_Fragment: Listas desplegables con la documentación.

Más adelante en el manual de usuario se detallará visualmente los diferentes Fragmentos y sus usos.

Como mencionamos anteriormente el paquete GUI no solo contiene la parte de la interfaz, sino que también encontraremos un paquete denominado “Tools” donde hemos almacenado todas las clases que proporcionaron ayuda a la parte interfaz, como lo son:

- BitmapLRUCache: Almacenamiento de imágenes.
- ExpandableListAdapter: Adaptador de listas expandibles.
- PieGraph, PieSlice y Utils: Creación de gráficos circulares.
- SimpleFileDialog: Manejo de archivos y carpetas.
- TouchImageView: Manejo de imágenes.

5.4.3. *Web Service*

Es el encargado de obtener y enviar información al Servidor además de ofrecer a la Interfaz un entorno que entienda y que sea sencillo de usar.

La clase principal de este paquete es **WebServiceTask,** clase que hereda de AsyncTask, una clase perteneciente a Android. **WebServiceTask** es la clase que se encarga de gestionar las llamadas y respuestas al Servidor por medio de un hilo independiente al principal, así no congela la aplicación mientras obtiene respuesta del Servicio Web.

Para esta clase hemos implementado llamadas GET, POST y PUT**.** Usaremos una u otra dependiendo la llamada que queramos hacer siguiendo siempre la lógica y la guía de las reglas de nuestro Servidor, ya que no podremos llamar a una URL que no esté definida.

Este paquete ha visto incrementado su número de clases debido a la necesidad de abstraer y encapsular las operaciones permitidas con el WebService de nuestra App.

Por esta razón tenemos dentro de nuestro paquete WS clases como **callerWS, WSquery y Tradutor**.

callerWS: Es el encargado de abstraer a la interfaz de la clase **WebServiceTask,** simplificando su uso simplemente modificando un atributo interno usado para crear la URL, el atributo path y otro atributo llamado wst que resulta ser de la clase **WebServiceTask.**

Para ejecutar el path creado en el wst, llamamos a uno de los métodos *Call, que se corresponden con una de los tres tipos de llamada que hemos definido en el

WebServiceTask (GET, POST y PUT), esto envía la petición al Servidor que devuelve una respuesta, la cual se trata para obtener un resultado en forma de String.

```
public class CallerWS {
        WSquery path;
        WebServiceTask wst;

        public CallerWS( ) {
                // TODO Auto-generated constructor stub
                this.path= new WSquery();

        }

        public WSquery getPath() {
                return path;
        }

        public void setPath(WSquery path) {
                this.path = path;
        }

        public String getCall(Activity frag,String token){

        wst = new WebServiceTask(WebServiceTask.GET_TASK, frag, "GETting data...",token);
        wst.execute(new String[] { this.path.getQuery() });
        String out=null;
        while(out==null){
                out=wst.getResponse();
        }
        return out;
        }
        public String postCall(Activity frag,String token){

        wst = new WebServiceTask(WebServiceTask.POST_TASK, frag, "POSTINGting data...",token);
        for(int i=0;i<this.path.getParamsNames().size();i++){
                wst.addNameValuePair(this.path.getParamsNames().get(i), this.path.getParamsValues().get(i));
        }
        wst.execute(new String[] {  this.path.getQuery() });
        String out=null;
        while(out==null){
                out=wst.getResponse();
        }
        return out;
        }
        public String putCall(Activity frag,String token){

        wst = new WebServiceTask(WebServiceTask.PUT_TASK, frag, "PUTTINGting data...",token);
        wst.setFileName(this.path.getFileLocal());
        wst.execute(new String[] {  this.path.getQuery() });
        String out=null;
        while(out==null){
                out=wst.getResponse();
        }
        return out;
        }

        public String getResponse(){
                return this.wst.getResponse();
        }
}
```

Figura 22. Implementación CallerWS

Traductor: Es la clase responsable de traducir el String obtenido por el **CallerWS** a un objeto que sea manejable en JAVA, por lo que hace uso de SIMPLE en caso de que obtengamos un XML o JACKSON en caso de que se trate de un JSON para realizar

Marshalling y obtener la clase de Esctructuras que corresponda a lo que habíamos pedido al Servidor.

```
public class Traductor {

	/*
	 * Clase que nos sirve para transformar un JSON en lo que queremos,
	 * Hay que notar que solo lo hara correctamente si invocamos el metodo
	 * que coincide con el STRING JSON que corresponde.
	 * Pej:
	 * Si llamamos para obtener una lista de paises tendremos que invocar
	 * el metodo getPaises, ya que cualquier otro no nos devolvera lo que
	 * queremos.
	 */
		String JSON;


	public String getJSON() {
			return JSON;
		}


		public void setJSON(String jSON) {
			JSON = jSON;
		}


	public Traductor(String json) {
		// TODO Auto-generated constructor stub
		this.JSON=json;
	}
	public UserWSType getUser() throws Exception{
		if(this.JSON.startsWith("<?xml")){
			Serializer serial = new Persister();
			UserWSType data = serial.read(UserWSType.class, this.JSON);
		return data;
		}else{
		Gson gson = new GsonBuilder().create();
	JsonParser parser = new JsonParser();
	JsonObject data = parser.parse(this.JSON).getAsJsonObject();
	return gson.fromJson(data, UserWSType.class);
		}
	}
	public NewSession getSession() throws Exception{
		if(this.JSON.contains("<?xml")){
			Serializer serial = new Persister();
			NewSession data = serial.read(NewSession.class, this.JSON);
		return data;
		}else{
			Gson gson = new GsonBuilder().create();
		JsonParser parser = new JsonParser();
		JsonObject data = parser.parse(this.JSON).getAsJsonObject();
		return gson.fromJson(data, NewSession.class);
		}
	}
	public CategoryWSType getCategoria() throws Exception{
		if(this.JSON.contains("<?xml")){
			Serializer serial = new Persister();
			CategoryWSType data = serial.read(CategoryWSType.class, this.JSON);
		return data;
		}else{
		Gson gson = new GsonBuilder().create();
	JsonParser parser = new JsonParser();
	JsonObject data = parser.parse(this.JSON).getAsJsonObject();
	return gson.fromJson(data, CategoryWSType.class);
		}
	}
	public CountryWSType getPais() throws Exception{
		if(this.JSON.contains("<?xml")){
			Serializer serial = new Persister();
			CountryWSType data = serial.read(CountryWSType.class, this.JSON);
		return data;
		}else{
		Gson gson = new GsonBuilder().create();
	JsonParser parser = new JsonParser();
	JsonObject data = parser.parse(this.JSON).getAsJsonObject();
```

Figura 23. Implementación Traductor

WSquery: esta clase es la que permite crear cualquier URL aceptada por el Servidor. Permite construir las URL por medio de métodos simples.

```
public class WSquery {
        /*
         * Clase que a traves de metodos nos construye cualquier URL que
         * acepta nuestro WS.
         * Ir contruyendo la URL por medio de los metodos para luego por
         * medio del caller llamar al WS y obtener la respuesta.
         */
        private ArrayList<String> paramsNames;
        private ArrayList<String> paramsValues;
        private String url="http://acr2.programame.com/ws/";
        private String login1="session?user=";
        private String login2="&password=";
        private String login3="&app=";
        private String app_id="2015-2015";
        private String query;
        private String str="?start=";
        private String size="&size=";
        private String fileLocal;
        private String fileWS;
        public enum type{
                allproblems,
                best,
                cat,
                code,
                comment,
                country,
                currentuser,
                institution,
                login,
                paths,
                problems,
                problem,
                ranking,
                submission,
                user,
                volume,
        }

        public WSquery() {
                // TODO Auto-generated constructor stub
                this.paramsNames= new ArrayList<String>();
                this.paramsValues= new ArrayList<String>();
                this.query=this.url;
        }
        public void addType(type tipo){
                this.query=this.query+tipo.toString()+"/";
        }
        public void addID(int id){
                this.query=this.query+Integer.toString(id)+"/";
        }
        public void addStartSize(int start, int size){
                this.query=this.query+this.str+Integer.toString(start)+this.size+Integer.toString(size)+"/";
        }
        public void addNumSubCat(int md){
                this.query=this.query+"?md="+Integer.toString(md)+"/";
        }
        public void addFree(String str){
                this.query=this.query+str+"/";
        }
        public String getQuery(){
                return this.query;
        }
        public void login(String usr, String pass){
                this.query=this.url+this.login1+usr+this.login2+pass+this.login3+this.app_id+"/";
        }
        public URI getURI() {
            return UriBuilder.fromUri(query).build();
        }
        public void addParam(String name, String value){
                this.paramsNames.add(name);
                this.paramsValues.add(value);
        }
```

Figura 24. Implementación WSquery

6. Manual del Usuario

A continuación presentamos el manual de la aplicación, junto con imágenes tanto de la aplicación como de la página web para comparar las funcionalidades y la navegación. Como requisito previo a la ejecución es necesario tener instalado como mínimo la versión "Jelly Bean", v4.0 de Android en el móvil para que la aplicación funcione correctamente con todos los *plugins* con los que se ha implementado.

6.1. Login

Tras iniciar la aplicación nos aparecerá la siguiente ventana que corresponde a la identificación del usuario.

Figura 25. Login

El usuario podrá entrar a la aplicación AceptaElRetoApp con su cuenta de AceptaElReto o crearse una nueva siguiendo el link "¡Registrate!".

6.2. Barra navegación

Nuestra barra de navegación está orientada en las funcionalidades principales que ofrece la página web AceptaElReto para sus usuarios, estas incluyen, la navegación entre los problemas, estadísticas, documentación, perfil de usuario y últimos envíos.

Figura 26. Barra de navegación

6.3. Inicio

Podemos empezar a explorar nuestra app a partir de esta ventana de inicio, que además de darnos una breve explicación acerca del juez online, nos da la opción de saltar hacia algún problema o usuario cuyo identificador (id) conozcamos.

Figura 27. Inicio

6.4. Perfil

Esta ventana nos proporciona toda la información de nuestra cuenta y nos permite modificar algunos campos si así lo deseamos. Debemos resaltar que ésta es la misma ventana, aunque con menos información debido a los permisos de usuario, que obtenemos al buscar un usuario desde la ventana inicio.

Figura 28. Perfil

6.5. Últimos envíos

Aquí encontraremos la información relevante acerca de nuestros más recientes envíos, como el resultado obtenido (Res) y nuestra posición en el ranking global del problema (Pos), entre otras.

Figura 29. Últimos envíos

6.6. Problemas

Nuestro buscador de problemas por tipo (Volúmenes y Categorías) es el mismo que el que encontramos en la página web. Es intuitivo y cómodo a la hora de buscar un problema.

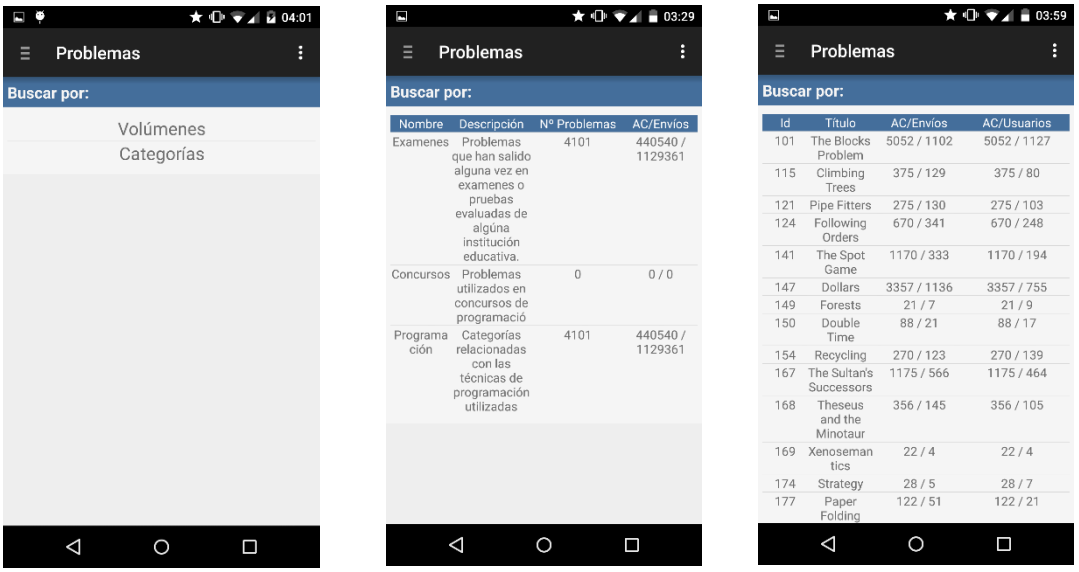


Figura 30. Lista problemas

6.6.1. *Problema*

Hemos creado un pequeño menú donde el usuario puede alternar entre ver el enunciado del problema, enviar una solución o ver las estadísticas generales del problema, facilitando la navegación y la usabilidad de la aplicación.

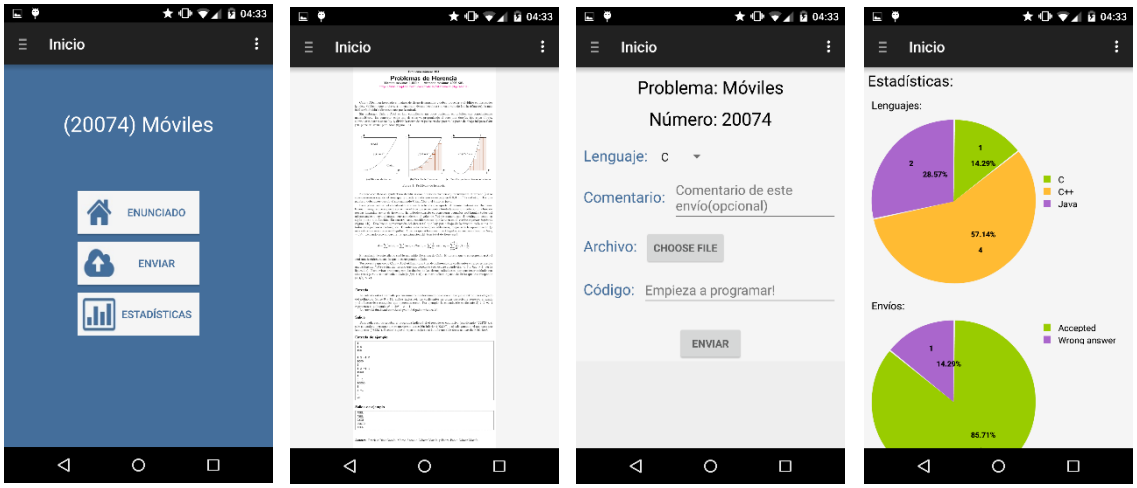
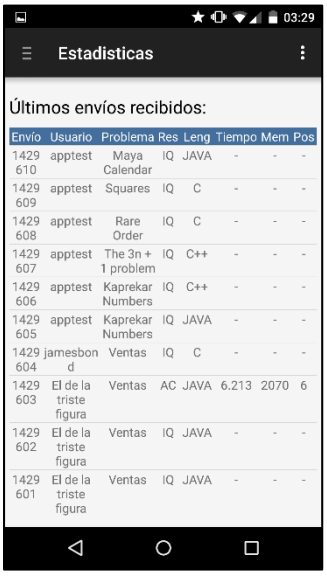


Figura 31. Problema

6.7. Estadísticas

Resumen de los últimos envíos recibidos por la comunidad de usuarios de Acepta el Reto.



Últimos envíos recibidos

Envío	Usuario	Problema	Resultado	Lenguaje	Tiempo	Memoria	Pos	Fecha
1429610	apptest	Maya Calendar (300)	IQ	Java	-	-	-	Hace 4 días
1429609	apptest	Squares (201)	IQ	C	-	-	-	Hace 6 días
1429608	apptest	Rare Order (200)	IQ	C	-	-	-	Hace 8 días
1429607	apptest	The 3n + 1 problem (100)	IQ	C++	-	-	-	Hace 9 días
1429606	apptest	Kaprekar Numbers (974)	IQ	C++	-	-	-	Hace 11 días

Figura 32. Estadísticas

6.8. Documentación

Aquí encontraremos la información pertinente a AceptaElReto, así como la respuesta a las preguntas más comunes y el significado de las abreviaciones encontradas en el apartado respuesta (Res) en nuestros envíos.

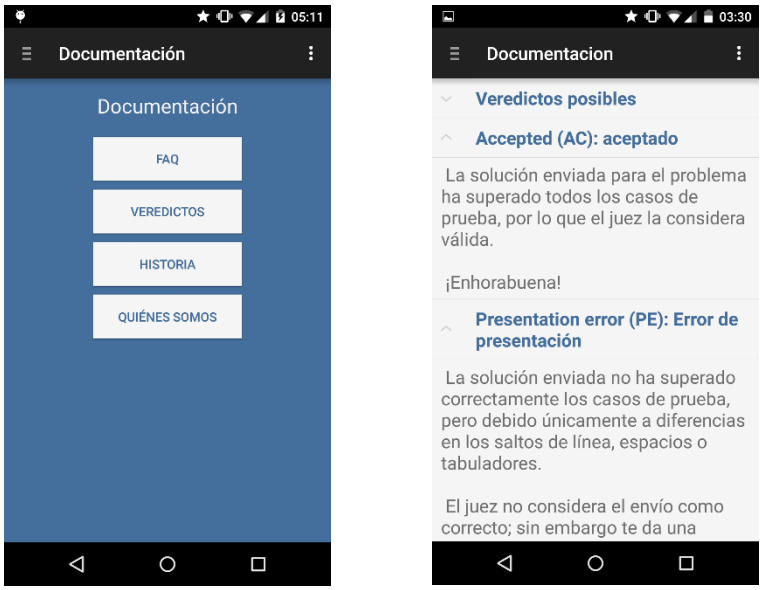


Figura 33. Documentación

7. Trabajo Futuro

Con el apoyo de los comentarios realizados en el código de nuestro proyecto y la estructuración que hemos elegido para el mismo, el cual se ha expuesto y profundizado en esta memoria, consideramos que será de fácil integración con la web AceptaElReto (ACR).

Creemos que nuestra aplicación AceptaElRetoApp podría ofrecerse como opción portable de ACR para dispositivos Android, y así llegar a un target de usuarios diverso.

Se podría implementar un nivel Administrador en la aplicación que tenga más funcionalidades que un usuario normal. Usando como base nuestro proyecto, solo se deberían ampliar algunas funcionalidades y ventanas que se encuentran deshabilitadas generalmente para los demás usuarios.

Otro objetivo a cumplir sería el trasladar esta aplicación a iOS para que los dispositivos Apple también puedan acceder a nuestra aplicación.

8. Bibliografía

- http://www.buenastareas.com/ensayos/Estado-Del-Arte-Android/30605696.html
- https://es.wikipedia.org/wiki/Android
- http://developer.android.com/develop/index.html
- http://developer.android.com/design/index.html
- Ben Smith. (2015). Beginning JSON. Apress.
- Frank Ableson, Robi Sen, Chris King. (2014). Android : guía para desarrolladores. Anaya Multimedia.
- Annuzzi J. Jr., Darcey L., Conder S. (2013). Introduction to Android Application Development, 4th Edition. Addison-Wesley Professional.
- Brian Hardy, Bill Phillips. (2013). Android Programming: The Big Nerd Ranch Guide. Big Nerd Ranch Guides